Enforcement of VO and Grid-level Policy with Standard POSIX Semantics

Open Science Grid Policy Technical Group

Markus Lorch
Virginia Tech

December 1, 2004
Revised January 19, 2005

1. Introduction

VOs and Grid Sites strive to define fine-grain, grid-level access control policies that protect their grid resources. These policies become more and more sophisticated and complex. Advanced management of access privileges may follow the role-based access control (RBAC) principle the structure of security policies by providing the organizing principle of a "role". A user is entitled to assume a (set of) roles. Authoritative entities to whom the user is known (e.g. entities managing VO membership) define the set of roles a particular user can assume. The user can, for each service request, select under what role this service request should be served. Policy administrators define RBAC policies which assign concrete access rights to each role.

In the Privilege Project, a joint project by US-CMS and US-ATLAS to provide advanced grid authorization mechanisms for OSG, the RBAC model has been adopted to manage access rights (privilege management). Access rights are to be granted based on a user's attributes that include VO membership, VO subgroup/division membership and selected roles that the entity may assume.

Access-control policies can be defined and implemented in a variety of ways. One popular approach is to define the policy in a high-level expressive language, such as XACML [ref OASIS]. Policies in such languages typically are applied by a Policy Decision Point (PDP) evaluating the policy with regard to a specific access request and available supplemental information (e.g., subject attributes). For the enforcement of the decision provided a Policy Enforcement Point (PEP) is required. The PEP may be implemented as a "door or gateway" through which all accesses to the service must be routed. Alternatively the service may be required to enforce all access decisions in its application code. These approaches become problematic if access to the service must be allowed through a number of different mechanisms or if the service code cannot be trusted to reliably perform appropriate enforcement of access decisions. These approaches also undermine or circumvent security semantics already present in the underlying execution environment (i.e., the operating system) by utilizing pool or service accounts that are endowed with elevated access rights.

An alternative approach is to provision the policy in the security semantics of the environment within which the service application will be executed. This approach of controlling the execution environment is sometimes referred to as sandboxing. Standard operating systems provide execution environments in the form of user accounts. The operating system semantics such as OS access rights to OS objects allow for the encoding of simple policy rules.

The advantages of this approach include:

- Security in depth through the utilization of high-performance, tested and trusted security semantics present in the underlying operating system
- No use of service or pool accounts – requests are served in individual execution environments to achieve isolation of requests from different users
- policy rules are always enforced, not only when the user or application accesses the system via grid layer mechanisms and "service doors" but also when legacy access mechanisms (e.g., POSIX file access) is performed or when the user is interactively connected to the resource (e.g., a local user)
- trusted service applications are not needed and user-provided (legacy) code can be securely executed
- existing operating system tools and mechanisms continue to work as designed and ease auditing and administrative tasks

The use of POSIX operating system semantics for the enforcement of high-level access control policies is frequently viewed as undesirable or impossible due to perceived limitation in expressiveness that may constrain the set of implementable policy rules.

In this document we will discuss an approach to leverage standard POSIX operating system mechanisms to reflect and enforce high-level policy rules. An example is given how the VO policy rules required by the U.S. CMS grid collaboration can be enforced through the novel use of existing operating system security semantics based on the POSIX recommendations. Implications of this approach with respect to the management of operating system user and group semantics and their use in cluster environments are also discussed.

2. Implementing VO Policies with POSIX OS Semantics

Note: focus more on dynamic assignment of groups

Discuss in more length the issues in cluster computers

The mechanism described here allows the continued use of legacy applications while improving security by reducing the amount of access rights a specific request is served (A first step toward least privilege access). The mechanism also improves accountability, confidentiality and protection, as requests by one user would be isolated from the requests of other users.

The VO Privilege Project sponsored by US CMS and US ATLAS provides authorization infrastructure components that, among other functionalities, enable the centralized and dynamic mapping of grid users to a set of local operating system qualifiers (e.g., local user account, local user group, set of supplemental local user groups) through an identity mapping service (GUMS). The policy that drives this mapping may take user-selected VO-membership and role attributes into account when making a mapping decision and thus allow the user to select a specific set of roles for a service request (independent from other service requests by the same user). Furthermore pluggable authorization modules are provided that allow grid services (e.g. compute gatekeeper, gridFTP, SRM storage services) to replace service-specific mapping mechanisms mechanisms that were constrained to a local decision on local user account and associated group accounts based on a (static) local user account mapping files and (static) entries in the local /etc/passwd and /etc/group.

To reflect the various roles a user may assume (and have requests serviced under) at the operating system the proposed mechanisms maps VO roles (VO and VO-subgroup membership membership can be called as a basic role) to operating system user groups. Basically the abstraction is as simple as:

Grid Identity (Certificate DN) --> Unix user
VO Role(s) --> Unix group(s)

The mapping to groups is based on the attributes (VOMS attributes) a user presents with the service request. The user can change these attributes for different service requests which will generate differing group assignments under which the request is served. The static assignment of a primary group and a set of supplemental groups to a user id via the /etc/passwd and /etc/group files is not taken into account. In fact this mechanism recommends that the users only statically allocated group is a so called “personal user group” as the primary group. This group is only assigned to this particular user and thus prevents users from unintentionally giving access rights to other system users through the (default) shared primary group. This practice of personal user groups is the default in many current Linux distributions.

The approach of dynamically setting the user’s group memberships for the particular request has many advantages, including:

- access rights for roles can be configured directly into the operating system mechanisms, the existing OS tools can be used (e.g. quotas)
- the user can dynamically select and act under different roles, even concurrently, the different roles are isolated against each other

- the user's actions are clearly traceable and connected via the common user account (in contrast to a single user using a set of role specific user accounts which are shared with other users that hold the same roles)
- users with the same roles (e.g. the same VO or sub-group membership) can easily share access to the same files, yet the file ownership and creator association is maintained
- OS tools like ps give meaningful information (in contrast to the use of shared accounts)

VO Role's may be complex and hierarchical. The POSIX group semantics describe a flat structure. One problem of this approach lies in mapping the hierarchical role structure in VOs to the flat group structure in Unix. The answer is to support only a few simultaneous roles and map these roles to individual, complementary POSIX groups. For example, a user "CN=Markus Lorch" may select to run a job as member of the VO "uscms" and in the uscms-role "production". On the resources this request will be mapped userid "mlorch", primary group "mlorch" (the personal group), supplemental groups "uscms-member", and "uscms-production".

Before we discuss the mechanisms further we provide a review of a few fundamental security functionalities:

1. A user (e.g., in BSD + Linux, not System V) can be signed up for up to 17 groups simultaneously in one session and thus combines the rights from all 17 groups without having to switch between groups. (1 primary group, 16 supplemental groups). The number of supplemental groups is a kernel compile-time parameter. The user does not have to be a "static" member of any of these groups as defined in /etc/groups. During the configuration of a users environment the system-calls initgroups, setgid and setuid can be used by the super user (or a process running with super user privileges, such as e.g., the Globus gatekeeper) to set the user and group ids to any (arbitrary) value.

2. File access permissions can be stated for multiple groups and users individually if file system access control lists are used (supported in all major Unix OSes e.g. Linux, Irix, Solaris, Aix) E.g., it is possible to have the following ACL:

Standard POSIX entries:	Additional POSIX ACL entries:
owner: mlorch, rw	group: uscms-member, r
group: mlorch, r	group: uscms-admin, rw
other : --	group: uscms-production, rw
	user: backup, r

3. Implementation Example

The following is a description of how this mechanism could be implemented at a site such as Fermilab:

3.1 Operating System User Allocation

At a site each user will have its own local user account, this can either be a standard account statically created and also used for interactive access, OR it can be a dynamic grid user account that is assigned upon first access. A pool of dynamic grid user accounts for OSG users on the order of about 10000 UIDs is set aside up front, e.g. UID 40000-49999. Once a grid user has been assigned a dynamic user account this assignment will become permanent. IF a dynamic user account is not being utilized for a lengthy period of time (months?) it will go through the standard manual deallocation procedure that is in place for standard accounts. Dynamic accounts cannot login interactively. Dynamic accounts also each have their own group (personal group) such that these accounts do not accidentally give each other access rights through a common default group. This also eliminates the need of modifying the /etc/password file once an account is allocated (the allocation happens externally by GUMS) as the primary group will never change no matter to whom it is allocated. In summary /etc/passwd and /etc/group will be primed with the 10K accounts on each machine. A site central GUMS server keeps track of the assignment of these ids to users based on DN or whole certificates for individuals.

3.2 Operating System Groups

For each Grid VO a set of unix groups are created, e.g. USCMS-member, USCMS-admin, USCMS-production on each resource. The group is assigned all the file rights that are shared among the VO members. INDIVIDUAL USER ACCOUNTS ARE NOT STATICALLY ENTERED INTO THESE GROUPS VIA /etc/group but rather the gatekeeper authorization process selects the set of groups that should be active for the current access

3.3 Mapping Process (general)

When a user requests grid access he will supply a VO/role attribute indicating what type of privileges will be required for this access and with what rights file access/file creation should take place. E.g. if a user requests access as USCMS VO member he will be mapped to USCMS-member (the default group for all USCMS access requests). If a user must perform administrative tasks for the USCMS VO he will request both the uscms-member and the uscms-admin group by selecting the "admin" role within the USCMS VO.

3.4 Mapping Process (cluster computer)

In the case of a cluster computer on which the gatekeeper hands off the job to the queuing system it gets a bit trickier but still doable. As the job itself will actually be executed on a system remote to the gatekeeper the gatekeeper process cannot easily influence the set of groups a user shall be mapped to for the specific process. However the startup of the remote job typically involves the setup of a user shell via rpc, ssh or some special daemon (as in lammpi). This establishment of the remote shell must be synced with the local shell at the gatekeeper that submitted the job to the queuing system. Possible approaches include installing a lean gatekeeper or sudo-process on the worker nodes that will only be accessible from the local machine and will set the local execution environment the same way the gatekeeper did it on the front-end machine, either through independently querying the identity mapping service (the worker node should have the user's delegated proxy certificate available to extract the attributes) or, even simpler, by re-using the mapping response that the front-end gatekeeper has gotten from the mapping server when the job was initially submitted. Dynamically updated NIS systems etc. cannot be used as several requests from the same user may be served concurrently, each with individual role assignments. Without a mapping/setuid mechanism on the worker node a user would be only mapped to his personal group and would not have the additional access rights afforded by the VO roles while running on the worker node. This may be sufficient for some environments where such access rights are not needed for the compute-process running on the worker node.

3.5 Requirements Summary

A scheme like this for a grid node in OSG may have requirements that are similar to these:

1. 10000 user IDs for dynamic accounts
2. 10000 group IDs for personal groups of dynamic accounts
3. Additional OS group IDs for the VO roles on the order of 5 * Number of supported VOs
4. all quotas are assigned via the group accounts (on the VO layer)
5. if the system is a cluster computer additional mechanisms need to be in place to assure that the worker nodes replicate the execution environment (the user and group mapping) of the front-end node